Classifying Abstracts of Biomedical Literature for Stem Cell Research

9th September 2015

DAVID ASHER

Supervisors: Mariana Neves, Prof. Dr. Ulf Leser
## Contents

1 Motivation .................................................. 4

2 Overview ................................................. 5
   2.1 Definition ........................................... 5
   2.2 Approach ............................................ 6

3 Feature Extraction ...................................... 6
   3.1 Bag Of Words (BOW) ............................... 7
   3.2 Feature Set Reduction ............................... 8
   3.3 Stemming and Lemmatizing ........................... 9
   3.4 Bigrams .............................................. 10
      3.4.1 Motivation ..................................... 10
      3.4.2 Induction Procedure ......................... 11
   3.5 Corpus .............................................. 13

4 Machine Learning ....................................... 13
   4.1 Support Vector Machines (SVM) ..................... 13
      4.1.1 Mathematical Formulation .................. 13
      4.1.2 Soft-Margin SVMs ............................ 14
      4.1.3 Nonlinear Decision Boundaries ............ 15
   4.2 Decision Trees ...................................... 16
      4.2.1 ID3-Algorithm ................................ 17
      4.2.2 Enhancement .................................. 18
   4.3 Nearest Neighbor Classification (k-NN) .......... 18
      4.3.1 Similarity Measures ......................... 19
      4.3.2 Nearest Neighbors with Information Gain 20
   4.4 Classifier Committees .............................. 21
   4.5 The Class Imbalance Problem ........................ 21
      4.5.1 Random Undersampling ....................... 21
      4.5.2 Near-Miss Sampling ......................... 22

5 Experiments .............................................. 22
   5.1 Evaluation Methods ................................ 23
   5.2 Setup .................................................. 24
      5.2.1 Baseline ....................................... 25
1 Motivation

Stem cells are biological cells that can differentiate into more specialized cells and have the ability to produce more stem cells through mitosis. It is widely believed, that if they are better understood, stem cell therapies can be developed, which would allow a dramatic improvement of the treatment of human diseases like cancer, Parkinson's, cardiac failures and many more. [18][19]

Due to their self-renewing and specializing property along with the widespread optimism among the biomedical research community, many scientific research papers on this topic are being published in biomedical journals.

The U.S. National Library of medicine provides the MEDLINE/PubMed database, which contains references to biomedical articles. It accommodates over 26 million references to articles that are published in over 5,500 journals with biomedical focus, which makes it the largest bibliographic database for biomedical literature.

Because of its sheer size and the not necessarily consistent terminology in biomedical sciences looking for bibliographical references only with keyword-based queries can be a very time consuming activity. In order to make this database more accessible for biologists, more sophisticated search methods have been developed. We briefly mention three approaches to facilitate the life of the researcher: searching with MeSH terms and the two projects GoPubMed and Caipirini.

To deal with the problem of ambiguous vocabulary in biomedical sciences, MeSH terms (Medical Subject Headings Terms) have been introduced. They form a hierarchically organized, expert-curated vocabulary of biomedical terms. Thus, we find for example in the hierarchical level, defined by the path ‘/anatomy/cells/stem cells’, terms like ‘adult stem cells’, ‘embryonic stem cells’, ‘fetal stem cells’, etc. Each newly added bibliographical item in MEDLINE/PubMed is associated with a set of major and minor MeSH terms, hence allowing to perform a search with a uniform vocabulary.

GoPubMed is another project with the aim of simplifying the work of the researcher. It allows the user to narrow the result of his MEDLINE/PubMed query with the help of four categories: the ‘who’, the ‘where’ and the ‘when’ categories let the user specify authors, location and date of publication of his desired references, whereas under the ‘what’ category the user finds the results of his query grouped according to the hierarchies of MeSH and GO (Gene Ontology, another controlled vocabulary for genes), which enables further curtailing of the result by deselecting unwanted terms.
However, not all of the articles, that are published in journals are necessarily interesting for biologists. Of the 104,332 articles in MEDLINE/PubMed which are associated with the MeSH term ‘stem cell’, we also find papers, that are dealing with funding, business opportunities, patents, moral aspects of stem cell research, legal situation in different countries, etc. MeSH terms do not help to filter these articles out, so it is desirable to have a tool which classifies articles into relevant and non-relevant for stem cell research.

The Caipirini project goes away from rigid and often too coarse-grained hierarchies and tries to adopt itself dynamically according to the researcher’s need. It does so by letting the user give examples of relevant and irrelevant documents via PubMed-IDs. Having the samples provided, a support vector regularization model is trained and a third list of PubMed-IDs can be sorted by their estimated probability to belong to the relevant set.

2 Overview

In this section we present a formal definition of our problem. We will approach the problem with methods from automatic text classification/categorization (TC) \cite{16,10,7}, which is a common problem in the field of information retrieval (IR) \cite{10} and machine learning (ML) \cite{12,4}. Following the definition, we outline the steps that are usually involved in constructing a classifier using techniques from TC and ML.

2.1 Definition

From now on we refer to the document space as $D$, whilst we denote with $C = \{-1,+1\}$ the set of possible document classes (‘irrelevant’ and ‘relevant’, respectively). Further, let $D_{\text{train}}$ be a training set of labeled documents $\{d_1, c_1, \ldots, d_\ell, c_\ell\}$, each $(d_i, c_i) \in D \times C$.

Given a training set $D_{\text{train}}$ and a supervised learning method $\Gamma$, our goal is to learn a classification function, which maps a document into a document class:

$$\gamma : D \rightarrow C$$

The classification function $\gamma$ can be thought as the result of the learning method applied to the training that, i.e. $\Gamma(D_{\text{train}}) = \gamma$. The objective for $\gamma$ is that it generalizes the training data well, i.e. it makes reasonably good predictions also for documents that are not element of training set.
2.2 Approach

Figure 2.1 shows the steps that are usually involved when building a text categorizer:

**Generation of the training set** Our classifier uses supervised machine learning algorithms. These algorithms require a training set of the form $D_{\text{train}}$, i.e. every document in the training has to be labeled with its corresponding document class. In Section 3.5 we describe how we derived this set.

**Feature Extraction** The next step is the conversion of the training set into a more suitable representation for machine learning algorithms. In particular we transform every document into a so called feature vector. Each dimension of the feature vector describes a numerical feature of the document. Section 3 gives an overview of all features we used in order to conduct our experiments.

**Machine Learning** In Section 4 we outline several state-of-the-art machine learning algorithms that we used to build our classification functions.

**Test** Finally, in Section 5 we describe in detail all the experiments we carried out. This description is followed by a discussion of the results.

3 Feature Extraction

Human written texts are highly unstructured and can vary greatly in length, which makes it difficult to handle them algorithmically. To get a more uniform representation of documents the vector space model [10] [16,7] has been proposed. The idea is to represent a...
document as a point in the vector space $\mathbb{R}^n$. In particular we derive from the training data a set of $n$ different features. Every document can then be transformed into its feature vector $\vec{d}_j = \langle w_{j1}, \ldots, w_{jn} \rangle$. Each dimension of this vector is associated with a feature. The values $w_{ji}$ are determined by a weighting scheme and should describe the degree of importance of the feature associated with dimension $i$ we attain to document $d_j$. In this section we present the feature induction methods and weighting schemes that we used for the representation of our documents. Further, in the last subsection we briefly describe the corpus from which we derive of the features.

3.1 Bag Of Words (BOW)

The bag of words model (BOW) is straightforward. Each document is regarded as multiset (bag) of words while ignoring the original position of a word in the document. That means for every document we merely consider how often a particular word occurs. This is known as term frequency and is defined for every document and for every word that occurs in the corpus as:

$$tf(t, d_j) = \text{number of times } t \text{ occurs in } d_j$$

If the vocabulary of our training sets consists of the set $\{t_1, \ldots, t_n\}$, we can represent the document $d_j$ as the vector $\vec{d}_j = \langle tf(t_1, d_j), \ldots, tf(t_n, d_j) \rangle$. This scheme, however, has the drawback that it has a clear bias towards longer documents. For correction it has been proposed to divide each document vector by the maximum term frequency that occurs in it. The bias can also be avoided by taking the boolean frequencies instead of the actual term frequencies, effectively regarding the document as a set rather than a bag.

The above mentioned schemes apply the same importance to each word in the corpus, which is clearly not the case. For example, words that occur in every document of our training collection are probably less discriminative than words that occur just in a few documents. For that reason the term frequency is often multiplied by the inverse document frequency:

$$idf(t) = \log \left( \frac{|D_{\text{train}}|}{|\{d \in D_{\text{train}} \mid t \text{ occurs in } d\}|} \right)$$

It equals zero if every document of the collection contains the word $t$ and the fewer documents contain $t$ the higher the value of $idf(t)$. 

7
3.2 Feature Set Reduction

Due to the nature of text categorization problems, the number of distinct features (words in BOW-representation) in the training set is much bigger than the amount of document available for training. With regard to the vector space model this means that the number of sample points is much smaller than the dimension of the vector space. With such a representation, due to statistical sparseness, machine learning algorithms are susceptible of attaining noisy words too much importance. This can lead to complex models which reproduce the training data very well, but have a poor overall performance for unseen data. A problem known as overfitting [10, 12]. The idea of feature set reduction is to remove noisy features, so that learning algorithms will never see them and thus remove the source of distraction. Reducing the number of features has the additional advantage that computationally expensive machine learning algorithms like $k$-NN scale better. We use the information gain of a feature as heuristic for noisiness.

Information Gain

Information gain [10, 12] is defined as the change of entropy when we observe a particular feature. Entropy of a random variable can be thought as the level of uncertainty of the possible outcome. In our context we are interested in the reduction of uncertainty of the document class. For a collection of documents $D$ the overall entropy amounts to [12]:

$$H(D) = -P('+') \log P('+') - P('-') \log P('-')$$

where $P('+)$ and $P('-)$ is the probability of a document to be of positively or negatively class, respectively. $H(D)$ is maximal when the classes are evenly distributed (maximum uncertainty) and minimal when the $D$ contains only one possible document class (no uncertainty). It is estimated by the relative frequencies of the document classes in the training set.

With this definition we define the information gain $IG$ for a document collection $D$ after observing a feature $f$ as [12] :

$$IG(D, f) = H(D) - \sum_{v \in values(f)} \frac{|D_v|}{|D|} H(D_v)$$
where \( \textit{values}(f) \) is the set of all possible values that the feature can attain and \( D_v \) is the subset of \( D \) so that the feature is of value \( v \). Low values of \( IG(D, f) \) indicate that feature \( f \) is probably noisy since it reduces the uncertainty of the document class only by a small amount.

### 3.3 Stemming and Lemmatizing

The English language uses inflection to express the grammatical category of a word. Moreover many words are derived from semantically similar ones with suffixes like ‘-tic’, ‘-ize’ or ‘-ation’. The goal of stemming and lemmatization \[10\] is to reduce inflected words to a common base form. Stemmers use a set of rules to cut of suffixes. It is applied on a single word without considering its grammatical context. As a consequence stemmer only produce heuristic results. Lemmatizers, on the other hand, also incorporate the part of speech of the word to be reduced. Ultimately, their aim is to reduce the word to a lemma of the predefined dictionary. The desired effect of stemming and lemmatization is similar to that of feature set reduction: Reducing statistical sparseness of the training data.

**BioLemmatizer**

In our experiments we tried to incorporarte BioLemmatizer which is an open source lemmatizer intended to be used in the domain of biomedical literature. The creators describe it as \[8\]:

“The BioLemmatizer [...] is tailored to the biological domain through integration of several published lexical resources related to molecular biology. It focuses on the inflectional morphology of English, including the plural form of nouns, the conjugations of verbs, and the comparative and superlative form of adjectives and adverbs. Given a word and its Part-of-Speech (POS) usage, the BioLemmatizer retrieves the lemma based on the use of a lexicon that covers an exhaustive list of inflected word forms and their corresponding lemmas in both general English and the biomedical domain, as well as a set of rules that generalize morphological transformations to heuristically handle words not encountered in the lexicon.”
3.4 Bigrams

Using bigrams as features for text categorization is not a novel approach. The well known RapidMiner \[11\] environment, for example, includes an generator for n-grams. However, to our best knowledge, little has been published on this area. Bekkerman and Allan \[2\] compared different approaches and developed an own induction method based on this study. We use their approach as base for our own induction procedure. In this section we describe how we derive new bigram features and briefly describe the differences of the methods.

3.4.1 Motivation

The BOW representation splits documents into their words and regards them as separated features, without taking into account the context of each word. Although, BOW has proven to be quite competitive, a fundamental drawback of this approach is the destruction of semantic relations that may exist between consecutive words. For instance, the phrase “Bill Gates” in the BOW representation of the document is represented by two single words: “Bill” and “Gates”. Given only the information that both of these words occur in a document, one may not conclude that the document is about computers, since it might be just as well about accounting or gardening. However, if the document representation provides information about the occurrence of the phrase “Bill Gates” the category of the document is clear.

Our analysis in the following is restricted solely on the impact of bigrams as additional features of our document representation. In contrary, one may suggest that phrases are often longer than two words, due to the existence of very long specialist terms in the biomedical domain. Take for example “human embryonic stem cell”. We give the following justifications: First, we observe that many specialist terms are assembled by stable bigrams themselves or at least have a sub term that is a stable bigram, e.g. the term “human embryonic stem cell” consists of the two stable phrases “human embryonic” and “stem cell”. Second, even if we include, say, four-grams we are likely to capture only a small fraction of all phrases whose discriminative power will be lost if split up into single words. So that the original demand, not to split phrases that belong together, is not satisfactorily met. On top, if we would be forced additionally to include trigrams and four-grams our document representation would become big and cumbersome.
**Algorithm 1** Adapted version of the bigram feature induction

**Require:**
- \(D_{\text{train}}\) – training set of documents
- \(W\) – set of all the distinct unigrams in \(D_{\text{train}}\)
- \(k_u\) – threshold on unigrams
- \(k_b\) – threshold on bigrams

**Ensure:**
- \(B\) – set of bigram-features

1: \(W' \leftarrow\) list of \(w \in W\) sorted by \(IG(w)\)
2: \(U \leftarrow\) set of \(k_u\) top ranked unigrams from \(W'\)
3: \(P \leftarrow \emptyset\)
4: **for all** \(d_i \in D_{\text{train}}\) **do**
5: \(LD_i \leftarrow\) list of unigrams in \(d_i\)
6: **for all** \(j = 2, \ldots, \) number of unigrams in \(LU_i\) **do**
7: \(w_{j1} \leftarrow LD_i[j-1]\)
8: \(w_{j2} \leftarrow LD_i[j]\)
9: **if** \(w_{j1} \in U\) **or** \(w_{j2} \in U\) **then**
10: \(P \leftarrow P \cup (w_{j1}, w_{j2})\)
11: \(F \leftarrow U \cup P\)
12: \(F' \leftarrow\) list of \(f \in F\) sorted by \(IG(f)\)
13: \(LB \leftarrow\) empty list
14: **for all** \(j = 1, \ldots,\) number of bigrams in \(F'\) **do**
15: \(b_j = (w_{j1}, w_{j2})\) be the \(j\)-th bigram from \(F'\)
16: **if** \(IG(b_j) > \max(IG(w_{j1}), IG(w_{j2}))\) **then**
17: **push** \(b_j\) to \(LB\)
18: \(B \leftarrow\) set of \(k_b\) top ranked bigrams from \(LB\)

**3.4.2 Induction Procedure**

Having an intuition why bigrams have the potential to improve the classification result, we address the question on how exactly to incorporate bigrams in our document representation. Should we use an representation that uses exclusively bigrams or should we use a representation that includes unigrams as well as bigrams? Bekkerman and Allan [2] suggest, that the first approach leads in most of the cases to a certain decrease of the classification result, while the second has the potential to improve results.

The main idea that lies on the basis of the bigram induction algorithm (Algorithm [1]) is to enrich the BOW representation only by bigrams that are more discriminative than already existing unigrams. As measure for discriminatory power for unigrams information gain (as defined in Section [3.2]) is used. The procedure is divided into two parts:
Creating bigram candidates (Lines 1–10) – A bigram is defined to be a candidate (member of set $P$) if one of its components is more discriminative than a parameter controlled threshold. To create the candidate set we first obtain a set $U$ of the top $k_u$ ranking unigrams. Subsequently, we go through each bigram that exists in our corpus and include it to our candidate set $P$ if one of its components is in $U$.

Discarding weak candidates (Lines 11–18) – Since we only want to include bigrams as features that are more discriminative than unigrams, we discard from our candidates all bigrams whose discriminatory power is less then the maximum of the involved unigrams. Therefore, we sort the set $U$ (the set of “good” unigrams, obtained in the first part) and all candidate bigrams by their information gain. After that, we go through each bigram candidate and push it to the result list $LB$ if the information gains of both components is less than the information gain of the bigram itself. Finally we return the $k_b$ top ranked of the so inducted bigrams. The last step exists merely to control the size of the bigram feature set.

Differences to the original induction procedure

The algorithm described in the section above is a slightly adopted version of the one presented by Bekkerman and Allen [2]. We briefly describe each modification that we made to the original algorithm along with the intended purpose that we tried to pursue with it.

The original algorithm uses mutual information with respect to a class as heuristic for the discriminative power of unigrams and bigrams respectively. This requires the algorithm to iterate over each class in order to obtain the set of high ranking unigrams $U$ and the set of high ranking bigrams $B$. In order to simplify the algorithm we used information gain which doesn't require us to provide a class.

The second adaption we made was changing the condition for a bigram in document to be included in the set $P$ of potential feature bigrams (Line 9). In the original version only two consecutive high ranking unigrams are included in $P$, whereas in our version it is sufficient that only one of the components has a good discriminative score. The reason for this is the observation that many biological terms often have parts that are numbers or Greek letters, which when standing alone most probably have a low information gain. For example we successfully included the high ranking bigram “tra 1”, which probably wouldn't have been included by using the unadapted version of the induction procedure.
3.5 Corpus

Löser et al. [9] examined 990 biomedical publications, published until the end of 2008, in order to find out how many publicly disclosed hESC-lines are available and to examine the impact of these publications. We used the abstracts of these articles as positive examples in our training set. However, due to the lack of a set of negatively labelled abstracts, we had to construct one by our own. We created a set of negatives by performing a search on the MEDLINE/PubMed database using the keyword “human embryonic stem cell” on all articles published until the end of 2008. From the result we dismissed all abstracts that also appear in our positive list.

4 Machine Learning

4.1 Support Vector Machines (SVM)

Assuming that the training data is linearly separable SVMs [16, 10, ?] try to find the separating linear hyperplane $\vec{w} \cdot \vec{x} + b = 0$ that has the largest margin between the plane and the nearest training samples. This allows us to classify an unseen document $d$ as $\text{sgn} \left( \vec{w} \cdot \vec{d} + b \right)$. In this Subsection we give a mathematical formulation of the problem and show how we can incorporate nonlinear decision boundaries as well.

4.1.1 Mathematical Formulation

The problem of finding a hyperplane with maximal margin can be described as quadratic optimization program. To see how this works we refer to Figure 4.1. The margin of the induced by the hyperplane depends only on a subset of the training samples, the so-called support vectors. These are exactly the samples that lie on the hyperplanes $H_1$ and $H_2$. They obey the following equations:

$$H_1: \vec{w} \cdot \vec{x} + b = +1$$

$$H_2: \vec{w} \cdot \vec{x} + b = -1$$

To successfully separate positive from negative samples $\vec{w}$ must be chosen such that:

$$\vec{w} \cdot \vec{d}_i + b \geq +1 \text{ for every } i \text{ with } c_i = +1$$

(4.1)
\[ \vec{w} \circ \vec{d}_i + b \leq -1 \] for every \( i \) with \( c_i = -1 \) \hspace{1cm} (4.2)

These constrains can be written more shortly as:

\[ c_i \left( \vec{w} \circ \vec{d}_i + b \right) - 1 \geq 0 \] for all \( i \)

The margin \( d \) between the hyperplanes \( H_1 \) and \( H_2 \) amounts to \( 2/|\vec{w}| \). Since we are interested in maximizing \( d \) we have to minimize \( |\vec{w}| \). Now we are able to formulate the problem of finding the separating hyperplane with the greatest margin as the following quadratic program [13]:

\[ \min_{\vec{w}, b} \frac{1}{2} |\vec{w}|^2 \quad \text{s.t.} \quad c_i \left( \vec{w} \circ \vec{d}_i + b \right) - 1 \geq 0 \] for all \( i \)

### 4.1.2 Soft-Margin SVMs

The formulation from above has the drawback that it doesn’t cope with fully linear separable problems. If the classes are not fully linearly separable, at least one of the constraints (4.1) or (4.2) will be violated. This is possible to remedy by introducing positive slack variables that relax the constraints and a penalty term in the original objective function, the following way:
\[ \mathbf{w} \cdot \mathbf{d}_i + b \geq 1 - \xi_i \text{ for every } i \text{ with } c_i = +1 \]
\[ \mathbf{w} \cdot \mathbf{d}_i + b \leq -1 + \xi_i \text{ for every } i \text{ with } c_i = -1 \]

which can be combined into a single constraint:
\[ c_i \left( \mathbf{w} \cdot \mathbf{d}_i + b \right) - 1 + \xi_i \geq 0 \text{ for all } i \]

Additionally we add a penalty term to our objective function. This leads to following quadratic program [13]:

\[
\min_{\mathbf{w}, b, \xi} \frac{1}{2} |\mathbf{w}|^2 + C \sum_{i=1}^{n} \xi_i \\
\text{s.t. } c_i \left( \mathbf{w} \cdot \mathbf{d}_i + b \right) - 1 + \xi_i \geq 0 \text{ for all } i \text{ and } \xi_i \geq 0 \text{ for all } i
\]

where \( C > 0 \) is a parameter that controls the impact of the penalty term. High values for \( C \) lead to a more accurate hyperplane but might result bad generalization. So a trade of value for this parameter has to be found.

### 4.1.3 Nonlinear Decision Boundaries

When the connection between two classes is highly nonlinear, also soft-margin SVMs fail to build a model that generalizes well. To introduce nonlinear decision boundaries into SVMs we first consider the dual version of the of the (primal) quadratic program for soft-margin SVMs. The version is derived by introducing Lagrange multipliers \( \alpha_i \) and taking advantage of the Karush–Kuhn–Tucker conditions [13]:

\[
\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j c_i c_j \left( \mathbf{d}_i \cdot \mathbf{d}_j \right) \\
\text{s.t. } \quad 0 \leq \alpha_i \leq C \text{ for all } i \text{ and } \sum_{i=1}^{n} \alpha_i c_i = 0
\]

\( \mathbf{w} \) is related to the the Lagrange multipliers \( \alpha_i \) the following way:
\[ \mathbf{w} = \sum_{i=1}^{n} \alpha_i c_i \mathbf{d}_i \]
Thus the separating hyperplane can be computed from an optimal solution as \[ 13 \]:

\[
\vec{w} \circ \vec{x} + b = \sum_{i=1}^{n} \alpha_i c_i \left( \vec{d}_i \circ \vec{x} \right) + b
\]

with

\[
b = \frac{1}{|S|} \sum_{s \in S} \left( c_s - \sum_{m \in S} \alpha_m c_m \left( \vec{d}_m \circ \vec{d}_s \right) \right) \quad \text{where} \quad S = \{ i \mid 0 < \alpha_i \leq C \}
\]

The Kernel Trick

In the derivation of the separating hyperplane from the optimal solution of the dual problem all training vectors \( \vec{d}_i \) exclusively appear in dot products. The idea of the kernel trick is to replace these products by a function that takes two vectors as arguments and non-linearly transforms into a different vector space them and returns the dot product of the transformed vector. Functions of this kind are known as kernel functions. Below we list the most commonly used kernels:

- polynomial kernels of degree \( d \)
  \[
k(\vec{v}, \vec{w}) = (\vec{v} \circ \vec{w} + r)^d
\]

- radial basis kernels which map input vectors into a infinite Hilbert space
  \[
k(\vec{v}, \vec{w}) = \exp(-\gamma |\vec{v} - \vec{w}|^2) \quad \text{with} \quad \gamma > 0
\]

- sigmoid kernels
  \[
k(\vec{v}, \vec{w}) = \tanh \left( \gamma (\vec{v} \circ \vec{w}) - r \right)
\]

4.2 Decision Trees

Internal nodes of decision trees \[ 12 \] (compare Figure 4.2) are labeled with tests on a particular feature. Edges departing from internal nodes are labeled with the possible outcomes of the test. Additionally, every leaf is labeled with a class. The classification for a document is started at the root. The edges are followed according the result of the internal node is
followed until a leaf is reached which eventually determines the class of the document. In the rest of this subsection we briefly describe the ID3-Algorithm and some enhancements.

### 4.2.1 ID3-Algorithm

The ID3 algorithm always produces decision trees that never misclassify an example of the training set. However, not all trees that are able to reproduce the training data achieve good classification results on unseen data. To counteract bad generalization of the training data the ID3 algorithm favours short and simple decision trees over big and complicated ones. More precisely, the ID3 algorithm tries to put tests on features that are likely to be very discriminative near the root and additionally guarantees that internal node tests on a particular feature tree appears at most once. The steps of the ID3-Algorithm can be summarized as follows:

1. If the training set contains only class then return leaf node labeled with this class.
2. If all features are already integrated in the tree then return leaf node labeled with the majority class of the training set.
3. Select from the unused feature the one with the highest information gain.
4. Create a new internal node test on this feature and partition the training set according the outcome of this test.
5. Recursively create subtrees for every partition.

The ID3-Algorithm can be viewed as a simple-to-complex hill-climbing strategy through the space of decision trees guided by the information gain.

We have yet to mentioned how to create the internal node tests. This depends on the form of the feature vectors. When feature weights are binary it is sufficient to test whether the feature is present in the document. Real valued feature weights can be incorporated by
using thresholds as depicted in Figure 4.2. A natural choice for the threshold is to consider every possible weight that a feature attains in the training set and choosing the one which maximizes the information gain.

4.2.2 Enhancement

The decision tree that is returned by the ID3-Algorithm is known to overfit the training data. To avoid this effect its successor, the C4.5-Algorithm [12], performs following post processing steps:

1. Convert the decision tree into a set of rules
2. Remove from each rule any precondition that results in an improvement of its estimated accuracy.
3. Sort the new rules by their estimated accuracy.

In the classification step the first rule that matches is applied. The accuracy of rule is estimated by computing the accuracy over the whole training set and then taking the lower bound of a hypothesis test for a fixed confidence interval.

4.3 Nearest Neighbor Classification (k-NN)

In contrast to SVMs and decision trees, nearest neighbor classifiers [12, 4] don't have a training phase. They store the whole training set instead of building a compact model. Therefore they are also called lazy learners. To classify an unlabeled document, the classification algorithm determines the “closest” documents in the training set to the instance to classify. These documents are called as the neighbors of the unlabeled document. The class memberships of the neighbors are used to assign a class to the unlabeled document. For example, in the most basic $k$-NN approach, the predicted category for the document is simply the majority class of the $k$ closest neighbors document in the training set. On each encounter the similarity between two documents has to be computed, whose computation time usually depends on the number of dimensions of the feature space. This is especially problematic in the area of text categorization where we have to deal with tens of thousands of features.

Nearest neighbor methods have been successfully applied in the area of text categorization. Han et. al. [15] present a $k$-NN classifier which is based on an adaption of cosine similarity. Ambert and Cohen [1] describe an approach which includes information gain to calculate
the closeness between documents. Their methods outperforms a SVM classifier in a categorization task for PPI (protein-protein interaction) related articles.

4.3.1 Similarity Measures

A natural first attempt for $k$-NN is to take the euclidean distance between two document vectors as a measure for dissimilarity. It can be easily converted into a similarity measurement by taking the building the reciprocal $1/d_{\text{euc}}$. The euclidean distance is defined as:

$$d_{\text{euc}}(\vec{v}, \vec{w}) = \sqrt{\sum_i (v_i - w_i)^2}$$  \hspace{1cm} (4.3)

This measurement, however has a major drawback [10]: the document vectors of two documents that are actually very similar, can vary greatly in their magnitude. For example (assuming that tf-idf is used as weighting scheme) two documents might share the same relative term frequency but one is document is twice the length of the other. As alternative it has been proposed to use the cosine of the angle spanned between two document vectors. This angle always lies between 0 and $\pi/2$ since all elements of the vector are positive, and it is independent of the vector magnitudes. The cosine similarity is defined as:

$$\text{sim}_{\text{cos}}(\vec{v}, \vec{w}) = \cos \theta = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_i (v_i \cdot w_i)}{\sqrt{\sum_i v_i^2} \sqrt{\sum_i w_i^2}}$$ \hspace{1cm} (4.4)

The denominator of the definition can be viewed as length-normalization for the two document vectors.

A natural heuristic for the similarity between two documents is the number of words they share. When the feature weights are binary the following definition gives exactly this number:

$$\text{sim}_{\text{ham}}(\vec{v}, \vec{w}) = \sum_{\forall i : (v_i = w_i = 1)} 1$$ \hspace{1cm} (4.5)

We also apply this measure to non-binary feature vectors, where we count the number of nonzero features whose absolute difference are below a predefined epsilon. In other words we changed the term below the sum to: $\forall i : (|v_i - w_i| < \epsilon \land v_i \neq 0 \land w_i \neq 0)$. 

19
4.3.2 Nearest Neighbors with Information Gain

The nearest neighbor classifier developed by Han et. al. [15] is based on the weighted cosine measure. It is defined as:

\[
\text{sim}_{\text{wcos}} \left( \vec{v}, \vec{w}, \vec{h} \right) = \frac{\sum_i (v_i \cdot h_i) (w_i \cdot h_i)}{\sqrt{\sum_i (v_i \cdot h_i)^2} \sqrt{\sum_i (w_i \cdot h_i)^2}}
\]  

Each component \( h_i \) of the vector \( \vec{h} \) describes how much importance we assign to the feature associated with the \( i \)-th dimension. In their original approach \( \vec{h} \) is obtained by a computationally expensive hill climbing algorithm. Having the results of Ambert and Cohen [1] in mind, we hypothesized that skipping the optimization algorithm for \( \vec{h} \) and associating \( h_i \) with the information gain of the feature associated with the \( i \)-th dimension would produce quite competitive results.

The similarity measure used by Ambert and Cohen [1] can be thought as a sophistication of \( \text{sim}_{\text{ham}}(\vec{v}, \vec{w}) \). It is defined as:

\[
\text{sim}_{\text{ig}} \left( \vec{v}, \vec{w}, \vec{h} \right) = \sum_{\forall i (v_i = w_i = 1)} h_i
\]

\( \vec{h} \) is again the information gain of the individual features. Contrary to the above approach, the predicted class is not obtained by mere majority voting. To predict a class for a document \( \vec{d} \) we determine its \( k \) closest neighbors \( N_k(\vec{d}) \). Further, for each class we sum up the similarity of the neighbors with the same class. The class \( c^* \) which is eventually assigned is the class with the highest score achieved in this manner.

\[
S(\vec{d}, \vec{h}, c) = \sum_{\vec{q} \in N_k(\vec{d}) \wedge \text{class}(\vec{q}) = c} \text{sim}_{\text{ig}} \left( \vec{d}, \vec{q}, \vec{h} \right)
\]

\[
c^*(\vec{d}, \vec{h}) = \text{argmax}_{c \in \mathcal{C}} S(\vec{d}, \vec{h}, c)
\]
4.4 Classifier Committees

For a task which requires expert knowledge, more then one expert often get better results than a single expert. This motivates the construction of classifier committees. Committees are known to work best when classifiers in the committee are as independent as possible [16]. We shall take this result into account by combining the three top ranking classifiers of the three different machine learning algorithms we use to conduct our experiments.

4.5 The Class Imbalance Problem

This section introduces the class imbalance problem [20, 14] and presents three different sampling methods to overcome problems affiliated with it. Machine learning algorithms are designed to minimize classification error and assume that the class distribution of unseen data is equal to the distribution of the training set [14]. This can have undesirable consequences for imbalanced training sets in which we are interested in the minority class. To see why this is the case, consider a highly skewed training set in which 99% of the samples are negative and 1% are positive. Based on the above assumptions, a rational thing for a machine learning algorithm to do, would be to create a trivial class classifier, that labels all inputs with the majority class. This, however, is unsatisfactory since with such a classifier we have no chance to detect an instance of a minority class document in which we are interested. Another way to look at the problem is to assign different costs for misclassification. In our case it is more costly to produce a false-negative than producing a false-positive. Our goal should be to build a classifier that produces as few false-negatives as possible even at the expense of producing more false-positives.

Class imbalances problems are very common in the area of text categorization since it is hard to obtain relevant documents for their need of the involvement of human experts.

4.5.1 Random Undersampling

Undersampling methods eliminate the prerequisite for the existence of the class imbalance problem by excluding from the training set samples that belong to the majority class. A simple way to balance out our training set is including each negative sample with the probability:

\[ p = \frac{N_{\text{pos}}}{N_{\text{neg}}} \]
4.5.2 Near-Miss Sampling

Every undersampling method comes with the risk of loosing information about the majority class. To overcome the risk of rejecting “important” samples, so called informed undersampling methods have been introduced. We describe two methods, namely NearMiss-1 and NearMiss-2 [6] that are based on nearest neighbor methods. Their objective is to include only those majority samples that lie close to minority samples.

In concrete terms, NearMiss-1-Sampling includes those sample from the majority class whose average distance to the three closest minority samples is the smallest. Whereas NearMiss-2-Sampling includes those samples from the majority class whose average distance to the three farthest minority samples is the smallest. In mathematical terms: Let for a vector \( \vec{v} \) that belongs to the majority class \( N_{\min}^3(\vec{v}) \) denote the set of the three closest samples of the minority class. Likewise let \( F_{\min}^3(\vec{v}) \) denote the three farthest minority vectors from \( \vec{v} \). Then for a distance function \( d(\vec{v}, \vec{w}) \) NearMiss-1-Sampling includes the majority vectors \( \vec{v} \) with the smallest score:

\[
S_{NM1}(\vec{v}) = \frac{1}{3} \sum_{\vec{w} \in N_{\min}^3(\vec{v})} d(\vec{v}, \vec{w})
\]

and NearMiss-2-Sampling includes the majority vectors \( \vec{v} \) with the smallest score:

\[
S_{NM2}(\vec{v}) = \frac{1}{3} \sum_{\vec{w} \in F_{\min}^3(\vec{v})} d(\vec{v}, \vec{w})
\]

5 Experiments

In this section we describe the experiment that we conducted. We combined different document representations and learning algorithms in five different rounds. The intention of each round is described in the following list:

1. In the first round we tried to find out if it is more beneficial to regard the words of a documents as a set or a bag. Based upon the result of this experiment we continued to use the representation that outperformed the other one.

2. The second round examined the the effect of including bigrams into our document representation. Since they have a positive impact on our results we kept including
them also in the rest of our experiments. If they had turned out to hurt classification performance we would have excluded them.

3. This round was conducted in order to see if we can benefit from applying stemmers or lemmatizers to our documents.

4. To investigate if we can improve classification result by classifier committees we combined several classifiers to a single one. To guaranty that the involved classifiers are as independent as possible we chose the best classifier of each of the three learning method from every experiment that we carried out so far.

5. In the last round we used the document representation and machine learning algorithm of the best classifier observed so far, to compare classification result of classifiers trained on unsampled and downsampled training sets, respectively.

### 5.1 Evaluation Methods

**Precision/Recall**

The most straightforward method \([4,12,10]\) to estimate the performance of a classifier is to partition the training set into three different sets: one set for training, one set for parameter tuning and one set for validation. Typically 66% are allocated for training and the remainder is evenly distributed into a tuning and a validation set. A trained and tuned classifier can then be tested on the validation set. For that we obtain the contingency table (see Figure 5.1) which consists of the number of correctly and incorrectly classified documents. Based on this table we define several measures.

<table>
<thead>
<tr>
<th></th>
<th>actual class</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
<td>Negative</td>
</tr>
<tr>
<td><strong>predicted</strong></td>
<td><strong>TP</strong> (true positives)</td>
<td><strong>FP</strong> (false positives)</td>
</tr>
<tr>
<td>class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Positive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td><strong>FN</strong> (false negatives)</td>
<td><strong>TN</strong> (true negatives)</td>
</tr>
</tbody>
</table>

Figure 5.1: Contingency table
The *precision* tells us the fraction of the retrieved documents, these are those which have been predicted as relevant, that are indeed relevant. It is defined as:

\[ P = \frac{TP}{TP + FP} \]

The *recall* is defined as the fraction of relevant document that have been retrieved:

\[ R = \frac{TP}{TP + FN} \]

Precision and recall are combined into one measure by taking the harmonic mean. The *F1-score* is defined as:

\[ F_1 = \frac{2 \cdot P \cdot R}{P + R} \]

**Cross-Validation**

We evaluated our classifiers by performing a series of cross-validation [12] studies. Cross validation is often used for small training sets. The idea of cross-validation is to randomly split the training set into \( n \) different non-overlapping subsets whilst assuring that each of these partitions have approximately the same class distribution of the training set. Afterwards for all possible \( n - 1 \)-combinations of the partition a classifier is trained. For each of the resulting \( n \) different classifiers we can obtain a contingency table by testing it on the partition that wasn't used for training this particular classifier.

**5.2 Setup**

In the first series of experiments we carried to find the best classifier and reduced feature set combination for three different BOW representation. The results served as baseline for the following experiments.

The following list describes the meaning of the abbreviations in the result tables.

- **euc** majority voting \( k \)-NN classifier with reciprocal euclidean distance \( d_{euc} \) as similarity measure
- **cos** majority voting \( k \)-NN classifier with dot product \( \text{sim}_{\text{cos}} \) as similarity measure
- **ham** majority voting \( k \)-NN classifier with “hamming similarity” \( \text{sim}_{\text{ham}} \) as measure; real numbers were treated as equal when their difference was below a predefined epsilon; the reason we included it was to see the benefit of ig1 over ham.
the $k$-NN classifier based on the weighted “hamming similarity” $\text{sim}_{\text{ig}}$ \ref{eq:hamming_similarity}.

majority voting $k$-NN classifier with weighed dot product $\text{sim}_{\text{wcos}}$ \ref{eq:cosine_similarity}.

support vector machine with linear kernel; we use the LIBSVM \cite{libsvm} library. The $C$ parameter was determined as suggested in the \cite{libsvm} by grid search.

support vector machine with radial basis kernel; as with the linear kernel the parameters $C$ and $\gamma$ were determined by grid search.

decision tree learner from the waffles \cite{waffles} library.

### 5.2.1 Baseline

The experiments were conducted using the C++ programming language. To extract the words from the documents we used the tokenizer from the boost-library which in its standard configuration is designed to extract tokens from linguistic sentences. We additionally converted all tokens to lowercase. Ranking word-features according to their information gain revealed that there were high ranking stop-words so that we decided to use a stop-word list to filter them out. Each classifier was tested with at first complete feature set. To examine the effect of reduction of the feature set we carried out tests with the 15000, 10000, 5000, 2500, 500, 250, 100, 50, 25, 10, 5, 1 features with the highest information gain. The performance of each classifier feature set combination was estimated with 10-fold-cross-validation. Considering the fact that documents of our positive class were about 16% longer than the ones of the negative class and that high ranking words appeared very seldom, we used boolean frequencies rather than the actual term frequency. Each feature vector of the tf-idf representation was normalized by dividing it through its euclidean length. The results of the experiments can be found in Table \ref{table:results1} and Table \ref{table:results2}. 

25
Table 1: $F_1$ values for the $k$-NN run for BOW (stop words removed) with binary feature vectors (left) and with cosine normalized tf-idf feature vectors (right)

As heuristic for the optimal value $k$ for all the the classifier via performed a series of cross-validation determining the F-score for all uneven integers and setting $k$ to the value where the highest score was achieved.

Contrary to our expectations the nearest neighbor classifier based on $\text{sim}_{\text{ig}}$ didn't outperform the other nearest neighbor methods on our data set. The “uninformed” similarity measures turned out to produce quite competitive results, with $\text{sim}_{\text{cos}}$ winning against all other methods on the binary representation and $\text{sim}_{\text{ham}}$ producing quite fair results. Comparing the results of $\text{sim}_{\text{cos}}$ and $\text{sim}_{\text{ham}}$, it seems that the subsequent length normalization of the cosine similarity on the (unnormalized) binary feature vectors has a positive effect on the classification result. Additionally it appears to be quite robust against the presence of many noisy features. Whereas the F-Scores of the euclidean classifier on tf-idf representation tends oscillates very much with producing trivial classifiers even on big feature sets.
We weren't able to reproduce the results from Ambert and Cohen ([1]). On our dataset a linear SVM significantly outperforms the informed nearest neighbor variants, with achieving its maximum F-Score on the 2500 highest ranking terms. The use of a non-linear kernel didn't have a great impact on the classification result throughout the whole series of experiments we conducted. In general our data set seems to be easy to classify considering the fact that the performance only drastically decreases when using less than the 25 top ranked features. The results of the tree classifier have to be considered as a lower bound since the DecisionTreeClassifier class of the waffles library doesn't implement any pruning techniques.

### 5.2.2 Bigram

The best classifiers of the baseline classifiers revealed that the representation of the feature vectors has no significant impact on the classification result. In fact binary feature representation outperformed the tf-idf representation. Based on this observation we decided to represent the bigrams as binary features as well. The results of the experiment can be found in Table 3. We also included a list of the ten top ranking bigrams in Table 4.
<table>
<thead>
<tr>
<th>#features</th>
<th>cos</th>
<th>svm</th>
<th>tre</th>
</tr>
</thead>
<tbody>
<tr>
<td>16897</td>
<td>0.75</td>
<td>0.84</td>
<td>0.78</td>
</tr>
<tr>
<td>15000</td>
<td>0.76</td>
<td>0.85</td>
<td>0.77</td>
</tr>
<tr>
<td>10000</td>
<td>0.78</td>
<td>0.86</td>
<td><strong>0.78</strong></td>
</tr>
<tr>
<td>5000</td>
<td>0.77</td>
<td>0.87</td>
<td>0.78</td>
</tr>
<tr>
<td>2500</td>
<td><strong>0.87</strong></td>
<td>0.77</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0.78</td>
<td>0.87</td>
<td>0.76</td>
</tr>
<tr>
<td>500</td>
<td>0.79</td>
<td>0.86</td>
<td>0.78</td>
</tr>
<tr>
<td>250</td>
<td>0.78</td>
<td>0.86</td>
<td>0.76</td>
</tr>
<tr>
<td>100</td>
<td>0.79</td>
<td>0.85</td>
<td>0.76</td>
</tr>
<tr>
<td>50</td>
<td><strong>0.80</strong></td>
<td>0.84</td>
<td>0.77</td>
</tr>
<tr>
<td>25</td>
<td>0.80</td>
<td>0.83</td>
<td>0.72</td>
</tr>
<tr>
<td>10</td>
<td>0.66</td>
<td>0.77</td>
<td>0.13</td>
</tr>
<tr>
<td>5</td>
<td>0.49</td>
<td>0.66</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.49</td>
<td>0.70</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: $F_1$ values for the BOW with binary representation enriched by bigrams

```
"human embryonic", "stem cell", "cell research", "stem es", "embryonic stem", "es cells", "tra 1", "adult stem", "potential conflicts", "disclosure potential"
```

Table 4: Ten top ranking bigrams

We started our induction procedure with the threshold value $k_u$ of 1000 for unigrams and didn't put a threshold for $k_b$ for bigrams. This lead to the induction of 65 new bigram features. We observed a significant increase of the classification result of all classifiers. The F-Score for the SVM classifier increased by four percentage points while the tree classifier and the nearest neighbor classifier with cosine similarity both saw a boost of five percentage points. It seems that documents with a biomedical domain can substantially benefit from the inclusion of bigrams, whereas this wasn't the case for newsgroup articles [2].

5.2.3 Stemming

The next series of experiment we carried out was to see whether stemmers and lemmatizers can improve the classification result. Unfortunately the BioLemmatizer library has proven to be computationally too expensive to be successfully included in our environment. Therefore we only concentrated solely on stemming. We used the same configuration as in the bigram experiments with the exception that we stemmed every token re-
turned by the tokenizer. As stemmer we used the Oleander Stemming Library which implements Porter’s stemming algorithm. The results of the run can be found in Table 5.

<table>
<thead>
<tr>
<th>#features</th>
<th>cos</th>
<th>svm</th>
<th>tre</th>
</tr>
</thead>
<tbody>
<tr>
<td>11703</td>
<td>0.74</td>
<td>0.84</td>
<td>0.78</td>
</tr>
<tr>
<td>10000</td>
<td>0.75</td>
<td>0.85</td>
<td>0.78</td>
</tr>
<tr>
<td>5000</td>
<td>0.77</td>
<td>0.86</td>
<td>0.78</td>
</tr>
<tr>
<td>2500</td>
<td>0.77</td>
<td>0.86</td>
<td>0.78</td>
</tr>
<tr>
<td>1000</td>
<td>0.80</td>
<td><strong>0.87</strong></td>
<td><strong>0.79</strong></td>
</tr>
<tr>
<td>500</td>
<td>0.79</td>
<td>0.86</td>
<td>0.79</td>
</tr>
<tr>
<td>250</td>
<td>0.77</td>
<td>0.86</td>
<td>0.77</td>
</tr>
<tr>
<td>100</td>
<td>0.79</td>
<td>0.84</td>
<td>0.76</td>
</tr>
<tr>
<td>50</td>
<td><strong>0.81</strong></td>
<td>0.85</td>
<td>0.78</td>
</tr>
<tr>
<td>25</td>
<td>0.80</td>
<td>0.84</td>
<td>0.72</td>
</tr>
<tr>
<td>10</td>
<td>0.60</td>
<td>0.79</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>0.52</td>
<td>0.70</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>0.79</td>
<td>0.70</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5: $F_1$ values for the BOW with binary representation enriched by bigrams with stemming enabled

Consistent with the finding as summarized by Sebastiani [16] we didn’t observe that the inclusion of a stemmer has a great impact on the classification result.

### 5.2.4 Committee

We also implemented a majority voting classifier committee consisting of the three top ranking classifiers:

1. SVM classifier with bigrams on the top 2500 features with a $F$-Score of 0.87
2. Tree Classifier with bigrams on the top 1000 features (stemming enabled) with a $F$-Score of 0.81
3. Nearest Neighbor Classifier with $\text{sim}_{\cos}$ measure on the top 50 features (stemming enabled) with a $F$-Score of 0.79

The resulting classifier had a $F$-Score of 0.85 therefore actually having a worse performance than the best classifier involved in the committee.
5.2.5 Downsampling

The last series of experiments was to examine the effect of downsampling the majority class. Therefore we split randomly partitioned the set of documents into a training and a test set, such that the test set contains 30% of the positive samples and 30% of the of the negative samples. The remaining samples were included in the training set. To test each sampling method we trained the top ranking SVM configuration one time on the unsampled training data and another time on the downsampled training set.

<table>
<thead>
<tr>
<th>Method</th>
<th>A</th>
<th>P</th>
<th>R</th>
<th>F₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsampled</td>
<td>0.88</td>
<td>0.82</td>
<td>0.80</td>
<td>0.81</td>
</tr>
<tr>
<td>Random</td>
<td>0.86</td>
<td>0.75</td>
<td>0.87</td>
<td>0.81</td>
</tr>
<tr>
<td>NearMiss-1</td>
<td>0.85</td>
<td>0.74</td>
<td>0.85</td>
<td>0.79</td>
</tr>
<tr>
<td>NearMiss-2</td>
<td>0.87</td>
<td>0.78</td>
<td>0.84</td>
<td>0.81</td>
</tr>
</tbody>
</table>

Table 6: Accuracy, precision, recall and F₁ values for different sampling techniques

NearMiss-1 sampling seems to hurt overall performance while random and NearMiss-2 sampling retain the same F-Score as the unsampled classifier. However, both random and NearMiss-2 sampling have a higher recall, which is desirable in many cases while the latter preserves a higher precision.

5.3 Summary of the results

The best classifier achieved a F-Score of 0.87. It was created by a linear SVM and incorporates additionally to the binary BOW representation of the training data a set of highly discriminative bigram features. It is possible to upscale the precision with random and NearMiss-2 sampling without hurting the overall F-Score while the latter maintains a higher precision. In general we seem to deal with a set which is easy to classify. The F-Score only collapses significantly when we include less than 25 features with highest information gain.

5.4 Outlook

Relevance Scores

Often not only the predicted class is of interest but also the certainty of the classification. While this is straightforward for nearest neighbor methods more effort has to be done to let
SVMs output also output the degree of certainty. Caiprini [17] for example trains a logistic regression SVM [?] and sorts unlabeled documents according the output.

**Additional Features**

Named Entities (NE) often give a significant performance boost in a biomedical domain. Text categorization systems that classify documents which are to protein-protein interactions for example often include a Named Entity Recognition (NER) component which reliable detects all proteins in document.

**References**


